

**Comprehensive Django Development**

**Course Number:** DGO-104
**Duration:** 5 days

**Overview**

Accelebrate's Comprehensive Django Development course teaches existing Python developers how to develop full-stack Web applications using the Django framework. Students learn the basics of creating simple models, views, and controllers, then move on to more advanced topics such as administration, session management, authentication, and automated testing.

**Note:**This course can be taught with Django 4.x or later on compatible versions of Python (as documented in the earliest release notes for your general Django release [1.11, 2.0, etc.] at <https://docs.djangoproject.com/en/2.0/releases/>).

**Prerequisites**

All students should have a working knowledge of Python, HTML5, and CSS.

**Materials**

All attendees receive comprehensive course materials covering all topics in the course.

**Software Needed on Each Student PC**

* Python 3.x installed (Anaconda bundle recommended)
* Django 1.x or 2.x framework
* Other Python modules installed
* Bootstrap framework
* A text editor or IDE (PyCharm Community Edition is an excellent free editor if you don’t have a strong preference)

**Objectives**

* Develop web sites based on content stored in a relational database management system
* Understand the model view controller (MVC) pattern and how it is implemented in Django
* Create Django templates for easy-to-modify views
* Map views to URLs
* Take advantage of the built-in Admin interface
* Provide HTML form processing
* Integrate automated tests with your code
* Take advantage of the built-in Admin interface
* Extend the template system
* Perform advanced database queries
* Implement documentation with Sphinx
* Create testable, reusable apps
* Migrate data
* Provide a RESTful interface to Django models
* Build responsive web applications that adapt dynamically to clients ranging from desktop computers with large screens to tablets and phones
* Cache requests to improve throughput

**Outline**

* Introduction
* Django overview
	+ Django is a framework
	+ Frameworks vs Packages
	+ What can it do?
	+ What are the alternatives?
* Getting Started with Django
	+ Sites and apps
	+ Shared configuration
	+ Minimal Django layout
	+ Built-in flexibility
* Usingcookiecutter
	+ Why cookiecutter?
	+ cookiecutter features
	+ Creating a project
	+ Creating an app
	+ Customizing cookiecutter
* Creating models with Django ORM
	+ Defining models
	+ Related objects
	+ SQL Migration
	+ App configuration
	+ Accessing models
* Login for Nothing and Admin for Free
	+ Setting up the admin user
	+ Running the admin site
	+ Tweaking the admin interface
	+ Changing the admin index page
* Basic Views (AKA Controllers)
	+ What is a view
	+ HttpResponse
	+ URL route configuration
	+ Shortcut: get\_object\_or\_404()
* Working with templates
	+ About templates
	+ Django template syntax
	+ Static files
	+ Loading templates
	+ The url tag
	+ Shortcut: template.render()
* Querying the Models
	+ QuerySets
	+ Field lookups
	+ Chaining filters
	+ Slicing QuerySets
	+ Related fields
	+ Q objects
* Enhancing Models
	+ Custom methods
	+ Complex relationships
	+ Overriding standard methods
* Forms
	+ Forms overview
	+ GET and POST
	+ The Form class
	+ Processing the form
	+ Widgets
	+ Validation
	+ Forms in templates
	+ Beyond the basics
* Debugging
	+ Printing to the console
	+ Django debug page
	+ Reading Django logs
	+ Django debug toolbar
* Class-based Views
	+ About generic views
	+ Types of generic views
	+ Default generic views
	+ Class-based generic views
	+ List and detail views
* User Authentication
	+ Authentication vs Authorization
	+ Configuring users
	+ Permissions
	+ Groups
* Session Management
	+ Enabling sessions
	+ Types of session backends
	+ Session cookies
	+ Accessing sessions from views
* Migrations
	+ Managed vs unmanaged tables
	+ Handling schema changes
	+ Using makemigrations and migrate
	+ Tweaking migrations
	+ Migrating data
* Advanced configuration
	+ Django settings architecture
	+ Typical configuration
	+ Using 12 factor config
	+ Security considerations
* Serving Data with a RESTful Interface
	+ The Django REST framework
	+ Serialization
	+ Requests and Responses
	+ Function-based views
	+ Class-based views
	+ What about django-tastypie?
* Static File Managment
	+ Types of static files
	+ Configuring access
	+ Namespacing
	+ Templates
	+ Deploying from outside Django
* Automated Testing
	+ Why create tests?
	+ When to create tests
	+ Using Django's test framework
	+ Using the test client
	+ Running tests
	+ Checking code coverage
* Using the Cache
	+ Types of caches
	+ Setting up the cache
	+ Per-site and per-view caching
	+ Low-level API
	+ Cache security
* Creating Reusable Apps
	+ Packaging
	+ Choosing an appropriate name
	+ Deploying
	+ Using setuptools
* Documentation with Sphinx
	+ Description of Sphinx
	+ reStructured Text
	+ Configuring sphinx
	+ Creating doc files
	+ Using autodoc
	+ Generating docs
* Conclusion